REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 9/15/95	3. REPORT TYPE AND DATES COVERED Semiannual 1/15/95 — 7/15/95

4. TITLE AND SUBTITLE
Dependable Distributed Scheduling in a Network of Workstations

5. FUNDING NUMBERS
N00014-94-1-0479

6. AUTHOR(S)
John Hammen, Arun Paramadhathil, James W. Cooley, Donald W. Tufts and Jien-Chung Lo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dept. Electrical & Computer Engineering
University of Rhode Island
Kingston, RI 02881

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Active Nodal Task Seeking (ANTS) concept uses a completely decentralized approach in which each computing node is an active member of the system. This not only avoids the possible critical region in favor of a dependable computing, but also provides an adaptable, distributed resource management that can explore the full potential in computational power of the system. This paper describes the dependable distributed scheduling of the ANTS approach in a network of workstations. Results of the implementation under the Sun OS and UDP/IP are reported. Experimental runs statistics showing the efficiency and effectiveness of the ANTS approach are also presented.

19950922 134

14. SUBJECT TERMS
Dependable distributed systems, fault-tolerant Computing, active mode operating system, ~~a~~ network of workstations.

15. NUMBER OF PAGES 17

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Dependable Distributed Scheduling in a Network of Workstations

John Hammen, Arun Paramadhathil, James W. Cooley,
Donald W. Tufts and Jien-Chung Lo
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805

Abstract

The Active Nodal Task Seeking (ANTS) concept uses a completely decentralized approach in which each computing node is an active member of the system. This not only avoids the possible critical region in favor of a dependable computing, but also provides an adaptable, distributed resource management that can explore the full potential in computational power of the system. This paper describes the dependable distributed scheduling of the ANTS approach in a network of workstations. Results of the implementation under the Sun OS and UDP/IP are reported. Experimental runs statistics showing the efficiency and effectiveness of the ANTS approach are also presented.

Keywords: distributed computing, decentralized distributed operations, multithreaded scheduling, network of workstations, UNIX daemon.

This work is supported by the Office of Naval Research under contract N00014-94-1-0479.

1 Introduction

The ANTS project [1] at the University of Rhode Island explores and evaluates the concept of Active Nodal Task Seeking (ANTS) as a way to efficiently design and implement a dependable, high-performance, and distributed computing environment. We have successfully implemented this concept in a network of workstations [2]. While our initial efforts have been concentrated on the dependability issues, we were also aware of the effectiveness and efficiency of the ANTS concept as a distributed scheduling method in a network of workstations.

A network of workstations can be commonly found in virtually any work place. Researchers have argued that the potential aggregated computational power of these workstations should be explored for coarse grained parallel computations of large scale problems. There have been many works on this subject. However, we shall concentrate ourselves on the ones with active mode operations, *i.e.* a computing node or a workstation actively seeks jobs to do rather than waiting idlely for job assignments. The first active mode distributed computing system that we are aware of is the Bell Laboratories' Safeguard [3]. Developed in 1970's for the Anti-Ballistic Missile (ABM) controls, Safeguard has many innovated features that we have incorporated into the ANTS concept. The most notable feature is the fully distributed task scheduling. The computing nodes in Safeguard are independently running their own tasks without any centralized mechanism. This feature is retained in ANTS [1] in favor of dependable computations.

The IBM AS/400 [4] is a shared memory distributed computing system. This system uses active mode operation plus special registers to guarantee an automatic load balancing. The shared main memory space is the centralized mechanism that will later become a dependability issue as well as a performance bottleneck. The Piranha [5] has been realized in a network of workstations and on CM-5 as well. The Piranha system assumed a centralized mechanism for job distribution in that the task queue that consists of all awaiting tasks is located in one centralized location. This obviously is a serious dependability issue and eventually a performance bottleneck. When hundreds or thousands of workstations are included in the computation of large scale problems, all communication to that one centralized location, for accessing the task queue, will certainly create a bottleneck. Phish [6] uses two levels of centralized job distribution mechanisms. At its top level, Phish assumed that each computing node will make inquiry to the centralized job queue in an interval of 30 seconds.

This will greatly reduce the effect of long communication delays and avoid possible bottlenecks. Once a workstation has gained control over a job from the centralized job queue, it creates a run-time *clearing house* for dispatching tasks to other workstations. This clearing house represents yet another centralized control mechanism. Also, this lower level scheduling policy, automatically rules out the possibility of contribution from other workstations to a given job, since the domain of participation is defined by the clearing house.

All these three active mode concepts are far from being considered dependable. The IBM AS/400 and Piranha depend on the inherent redundancy of any parallel or distributed computing system to provide potential dependability. Phish [6] claim that the fault-tolerance is guaranteed since redundant copies of a job exist in both top level and lower level centralized mechanism. However, the fact remains that no practical fault tolerant techniques have been proposed for these machines. On the other hand, many fault-tolerant distributed or parallel systems suffer from significant performance lost due to the inclusion of fault tolerance capabilities.

The ANTS concept was developed [1] to achieve both dependability and high performance. We assumed that each computing node or workstation maintains its own task queue of all awaiting tasks in the system. New tasks can be spawned easily by adding them to the task queue. The coherency of the distributed task queues are maintained by means of inter-node communications. While these scheduling methods ensure high-performance, they also provide enough redundancy for dependable computing. Simple error detecting and recovering techniques can be easily incorporated into the distributed kernel with minimum performance lost. The experimental results will show that a near linear speed up is likely when the average task execution time is much larger than the average communication delay. We will further show that the inclusion of fault tolerant techniques account for less than 5% performance lost.

2 ANTS Distributed System

Figure 1 shows the underlying hardware configuration of this study: a network of workstations. Specifically, we implement the ANTS concept on UNIX workstations that are interconnected via a 10Mb/s Ethernet. We caution here that this is merely one of the many possible hardware configurations for our ANTS concept.

In an ANTS system, an idle computing node will seek out the information it needs about pending, high priority work from partly prescribed, partly updated task tables, instead of waiting for other computing nodes to send it a job to do. Naturally, predecessor/successor conditions like those in a data flow architecture must be checked by the node. The advantage of this concept is that a true distributed system is guaranteed. All programs, including operating system and applications, can be executed distributively.

We define the term high-performance as the capability to fully explore the potential computational power. This is a suitable definition for scalable environment where computing nodes can be added to gain higher performance. ANTS can adapt itself to accommodate the increase in the number of computing nodes without any hardware or software modification. For the fault-tolerant version, the ANTS can add or delete computing nodes during run-time without interfering with the current running jobs. In locally connected system such as a system with several workstations interconnected via a local area network, this attribute of ANTS is very useful. Additional workstations can be added to obtain higher performance. Also, during run-time, some workstations that are heavily loaded with local jobs can stop their ANTS operation for the collaborated jobs. These workstations can easily re-enter the ANTS operation by again actively seeking out the next available tasks.

2.1 Job Distribution and Task Queues

When an ANTS job initiates, all related information such as tasks and data dependencies are broadcast to all ANTS computing nodes. A computing node, upon receiving the information, will update its own local task queue for this new job. A computing node, when idle, will pick up the high priority task from its own task queue for execution. There are several criteria for the task selection:

1. In a real-time environment, priority associated with each task will be considered first. Priorities of tasks are updated regularly according to the present deadline situation.

2. The second criterion is for an ANTS node to pick a task whose predecessor process, or at least one of them, has been executed by the same node. By doing so, we avoid the additional data transfer time since the predecessor data, or at least part of the needed data, is available locally.

3. A task queue may consists of thousands of awaiting tasks. Searching though all entries for an optimal match may be time consuming. Therefore, a user can define a lookahead

threshold which limits the depth of search by the kernel. The search depth usually directly relates to the algorithm itself.

2.2 Conflict and Coherency

A centralized task distribution can avoid the problem of conflict and coherency since the centralized mechanism handles requests sequentially. In ANTS, we need to address these problems due to the distributed nature. A conflict may occur when two or more computing nodes pick up the same task for execution. Each computing node will broadcast the message claiming the ownership of the task and then will start the execution immediately after. When other nodes receive the message, they will realize that they have picked the same task. We use a simple timestamp method to help resolve the conflict. A timestamp is associated with each task ownership message. By comparing the timestamps, a computing node will decide whether to continue the execution or to abandon the execution. In the worst case, a nearly completed task may be abandoned.

The frequency of conflicts can be greatly reduced by the following scheme. When a new job is initiated, not all computing nodes receive the tasks in the same order. For example, a new job with tasks 1 to 12 as shown in Figure 2. When this new job is initiated into local task queues of computing nodes, some may receive tasks in a sequence of 1, 2, 3, 4, 5, 6, 7, 8, etc., while others may receive tasks in a sequence of 3, 4, 1, 2, 7, 8, 5, 6, etc. We note that tasks 1 and 2 and tasks 3 and 4 are completely independent. By shuffling their orders in the sequence of tasks will not change the computational outcome, but will greatly reduce the possibility of a conflict. In the above example, half of the computing nodes may compete for task 1 while the other half compete for task 3. Of course the possibility of a conflict can be further reduced by a further diversity in tasks sequencing.

The coherency of local task queues cannot be guaranteed in an ANTS system. Because the updating message arrive at destination with with arbitrary communication delay. There are several factors that constitute a communication delay, namely: the media, the protocol, the possibility of an on-going communication, the local peripheral response delay, etc. We may say that although broadcasting is used to announce the ownership of a task, not all computing nodes receive the message and update their queue at the exact same time. However, we realized that this may not be a problem at all for the ANTS operations. The reason is that such inconsistency in local task queue contents does not significantly affect the performance nor the dependability of the system. Let us use the example shown in Figure 2 to illustrate this fact. When computing node A picks up and executes task 1, it broadcast the message to all other nodes. Node B at the same time picks up and executes task 2, it also broadcast the message to all other nodes. If node B does not update its local task queue in time when node A finishes task 1, in the worst case, node B may compete with node A for task 5. In other words, no deadlock or starvation problem may arise due to the inconsistency of the local task queues.

2.3 Communication Delays

Even though we consider coarse grain applications for a network of workstations, the communication delays still play an important role. For example, in Piranha [5], a task duration of 1 second is chosen for a reasonable speed up curve. In our initial implementation, we also find the a long averaging execution time of tasks enables a near linear speed-up curve. Of course, the numerical answer to what is the proper averaging execution time of tasks is directly related to the average communication delay of the underlying network. Nonetheless, we may conclude that it is beneficial to reduce the frequency of communication. We note that in ANTS all communications are broadcastings, but, in an Ethernet the two are equivalent.

Therefore, an important implementation goal is to reduce the frequency of communication. Besides job initiations, typical communications between ANTS nodes are:

1. Ownership message: A node broadcasts message to claim the ownership of a task. This happens immediately after an idle node picks up a new task to be executed.

2. Completion message: Upon the completion of the local task, a node broadcasts message to inform the completion.

3. Data request message: When a task has been acquired by a node and part of the data needed by this task were not available locally, a node broadcasts message to requests the data.

4. Data transfer message: A node sends out data in reply to the data request message.

5. Add Task message: Some task may request that new tasks being spawned. In that case, the node that is executing the task broadcasts this message to all other nodes for

updating the local task queues.

In order to reduce the frequency of communications, we have established the following schemes:

1. When a node picks up a new task, this node is either previously idle due to the lack of tasks in the local task queue, or has just completed a task. The former situation usually occurs only during initiation or during a shortage of tasks to be executed. The latter cases occur more often. In other words, we may say that a node usually picks up a new task when it finishes an old task. Therefore, we combine the ownership message with completion message in one broadcast effort.

   This policy also help reinforce a previous policy where a node will try to pick a new task whose data requirement can be satisfied locally. When a task is completed, no other node is aware of this fact except the local node until the local node picks up a new task and broadcasts the message. The possibility of conflict in claiming ownership of a new task is also reduced.

2. The data request message is also combined with an ownership and completion message. The only time a node is requesting outside data is when the new task calls for data not available locally. Obviously this occurs only when a new task is picked.

3. The new task message due to the spawning of new tasks from an old task can also be combined with the above message. There is a minimum performance lost since some task may request for spawning new tasks in the middle of its execution. Here, we will postpone that action until the completion of the old task.

Therefore, effectively, only three types of communication is required: *job initiation* (JI), *ownership, completion, data request, add task* (OCDA), and *data transfer* (DT). The JI messages is a one-time effort and occur only during initiation of a new job. The number of needed OCDA messages for a job is about the same as the number of tasks in that job. The frequency of DT depends on the data dependency of the algorithm. In fact, a carefully designed ANTS-executable algorithm should take this into consideration for a high-performance computation.

3 Designing Algorithms for ANTS

The ANTS concept provides a convenient way to distributed computing. The kernel or the distributed operating system itself already envisions several performance improvement features. Further improvement on performance is also likely by algorithms that are designed specifically with the ANTS operating principal in mind. In particular, granularity and inter-task communication frequency are the two major factors to be considered.

3.1 C Simulator

A simulator was written in the C programming language to run on a PC or on workstations. It is being used to test various strategies for the implementation of subroutines for computationally intensive algorithms. The simulator permits the inclusion of timing estimates of data input, internode communication, task time, and task precedence in terms of tables. Several versions were used to try several ways of having nodes select tasks. The most recent examines predecessors and estimates the communication time. A further enhancement being considered is to let the test look ahead as well as back to reduce internode communication. Doing this manually showed some improvements in the FFT algorithms.

3.2 Computational Algorithms

We are considering algorithms used in large compute-intensive applications. We have programmed and tested several subroutines for FFT's, QR algorithms, and L U decomposition. These dominate large applications such as finite element methods, large-scale fluid flow problems, and digital signal processing. The names and functions of our subroutines follow those used in Matlab[2]. This will permit their use in Matlab Mex files and make them accessible in Matlab programs.

3.2.1 FFT

Several ways of decomposing the radix 2 FFT algorithms were designed and run on the simulator. One of them is being implemented on a network of workstations. The algorithm is being generalized for powers of primes. Subroutines for powers of primes can be used for

[2]Matlab is a registered trademark of Mathworks Inc.

Fourier transforms of multiple arrays and as parts of more general mutually prime factor programs for arbitrary transform size N.

3.2.2 Linear Equation Solvers

The basic algorithm of many linear equation programs is the L U decomposition. Given the problem of solving $A \cdot x = y$ for x, one computes lower and upper triangular matrices L and U, respectively, such that $A = L \cdot U$. Since this requires n^3 operations, the task grows rapidly with problem size. The L and U may be used repeatedly in solving for many y's. They also give the matrix inverse and the determinant. The L and U are also very useful in some methods for very large eigenvalue computations.

3.3 QR Algorithms

Given a rectangular $m \times n$ matrix A with $m \geq n$, this algorithm uses Householder elimination to determine an $m \times n$ matrix Q and an $m \times n$ matrix R such that $A = Q \cdot R$ and the elements of R satisfy $r_{ij} = 0$ for $i \leq j$. This can be used to solve linear equations for a best least squares solution where there are more equations than unknowns. The elimination method will also be used in the eigenvalue subroutines.

3.4 Other Algorithms

We plan to write a number of FFT subroutines for special cases such real data, cosine transforms, etc. More linear algebra ANTS subroutines like those in Matlab are being planned. In particular, Eigenvalue computation and Singular Value Decomposition (SVD) are being studied.

4 The Experimental Implementation of ANTS

The ANTS prototype was tested on Sun Sparc workstations running Sun OS. The ANTS communication mechanism was layered on top of the Internet Protocols and the intercommunication network used was an Ethernet LAN running at 10 Mbits per second. The ANTS code was written in C++ and compiled using the GNU C++ compiler. The test application program was emulated using timing loops and an inter task data transfer size of 1.5KB was assumed. Predecessor and successor relationships of tasks are either generated randomly or are assumed to resemble that of the FFT computation. In addition, some randomly generated tasks with randomly generated data dependency relationships are also used. Due to the limitation in the number of available nodes, the test runs were limited to a maximum of nine nodes. Tests were done for three different *Task Slice Time* factors and a comparison of performance with and without fault tolerance was also made to study the performance trade off implications of using the ANTS Integrity Maintenance Protocols.

4.1 Fault Tolerant Features Verifications

The concurrent error detecting mechanism and other integrity maintenance protocols were tested using simulated faults. The tests included cutting off the power supply to nodes or to parts of the hardware, intentional triggering of faults by transmission of invalid data over the bus and simulation of memory parity errors. Isolated and multiple faults were simulated to fully test the correctness of the monitoring scheme. The observed results conformed to the specified requirements in that each fault free node reaches an accurate diagnosis of the fault conditions of the remaining nodes, without any restriction being placed on the number of faulty nodes or fault patterns.

The fault tolerant mechanism always correctly reconfigured the network in all the test cases. Under extreme load conditions, it took the ANTS system (with 9 nodes) a maximum time of one minute to reconfigure itself to a stable state. Under nominal conditions of load and faults the mean reconfiguration time was about 10 seconds. The reconfiguration time is the time interval between the occurring of a fault and the time the network becomes fully operational again, *i.e.*, the faulty nodes have been moved to the yellow group and available ANT nodes at the yellow group are re-enlisted to the green group. The reconfiguration time is related to the number of faults occurring at the same time and to the communication network latency.

Tables 1 and 2 show the typical reconfiguration times observed for the *isolated faults* and *chained faults* cases, respectively. The rather irregular variations in the reconfiguration time is due to the fact that, the monitoring protocols share the communication bus with the rest of the system. In such a situation the reconfiguration time is affected to a large extent by the application tasks that are running. The number of simultaneous or near simultaneous faults were limited to 6 in all the test cases. With more than 6 simultaneous faults and at high load levels the communication bottleneck was found to cause the system to become unpredictable.

4.2 ANTS Performance without Fault Tolerant Features

Figure 3 shows the performance figures obtained from the test results. During these test runs, the AFDRP or the Fault Tolerance Monitor is disabled. The modulation in programming ensures that these results are obtained with no interference from the disabled modules. The following inferences can be drawn from the results:

First, as is to be expected, the speed up factor is not linear. The speedup obtained gradually decreases with the addition of more nodes. The decrease in speed up is more pronounced with an increase in the number of nodes beyond 6. This effect is due to the traffic bottleneck imposed by the single communication channel. Addition of more communication channels or an increase in bandwidth of the communication media is necessary to achieve a better performance with more nodes in the system.

Another noticeable aspect of the performance figures is the higher linearity in speed up with higher Time Slice factors, *i.e.*, task execution time. This is evidently due to the fact that less overhead is involved with larger task fragments. The time slice factor will however have to be limited to a reasonable value to facilitate faster trapping of faulty nodes If the time slice factor were to be too large, a faulty node could unrecoverably damage the cooperative processing environment of the ANT system.

A third prominent feature is that with a large number of nodes in the system the speed up tapers of to an almost constant value. This value is a function of the total available bandwidth of the communication medium. This compares well with other distributed computing systems where beyond a certain point the speed up curve inverts and results in worsening performance [7]. This is due to the fact that with more nodes there are more messages that contend for the communication medium. In ANTS the communication traffic is not adversely affected by the number of nodes in the system.

We also observe that the average communication delay is about 200 to 250ms. There is a strong relationship between the average task execution time, the average communication delay, and the speed up factor. Of course, when a high speed network is available, a shorter task execution time can be used.

4.3 ANTS Performance with Built-In Fault Tolerant Features

We then record the performance figures of ANTS with AFDRP enabled. The test run results are shown in Figure 4. Figure 4 clearly shows that one of the major design goals has been achieved. The ANTS is designed to exploit the inherent fault tolerant features of the distributed system. We expect that a successful implementation of ANTS should have a minimum impact on the system performance. Test runs yielded a figure of 5% for the performance degradation due to the addition of fault tolerance. We believe that this low figure is a strong evidence to support that the inherent fault tolerance has been well utilized.

We have shown previously in [1] that an ANTS system can achieve an extremely long MTTF. Part of this MTTF improvement is due to the re-enlistment of ANT nodes with transient type failures. We note that the ratio of transient faults to all faults depends to a large extent on environmental conditions. In a documented case [8], the transient faults account for 20% to 33% in low earth orbit conditions. This clearly brings out the effectiveness of the recovery process of the ANTS fault tolerant mechanism. The penalty paid for obtaining such an improvement in MTTF is a minimal trade-off in performance, as shown in Fig 10. The loss of performance is a function of the number of nodes in the system and the mean performance degradation with 4 and 8 nodes was 2% and 5%, respectively. This is relatively small compared to the magnitude by which the recovery and re-enlistment process extends the useful life of the system.

5 Conclusions

We have presented in this paper a summary of the ANTS task scheduling policies and some experimental results. We clearly show that the major design goal has been achieved in that the high-performance and dependability can be achieved at the same time. Of course, the fault tolerant features discussed here do not include the cost for detecting or correcting data manipulation errors. To that end, the cost of checking may be much higher than reported here, since task duplication, triplication, or alike must be used. However, we also should point out that not all application require such a restricted data manipulation error detection/correction. In some cases, reasonableness checking as suggested in [1] can be used. Our initial design goal has been aimed at the achieving of an extremely long MTTF [1]. The results presented here show that this goal can be achieved as predicted.

References

[1] J. C. Lo, D. W. Tufts, and J. W. Cooley, "Active Nodal Task Seeking (ANTS): an approach to high-performance, ultra-dependable computing," *IEEE Trans. Aerospace and Electronic Syst.*, vol. 31, pp. 987–997, July 1995.

[2] P. Dominic-Savio, *Design of the ANTS Kernel to Implement a Fault Tolerant Dsitributed Computing System.* M.S. Thesis, University of Rhode Island, 1994.

[3] AT&T, "Safeguard supplement," *The Bell System Tech. J.*, 1975.

[4] J. E. Bahr, S. B. Levenstein, L. A. McMahon, J. T. Mullins, and A. H. Wottreng, "Architecture, design, and performance of Application System/400 (AS/400) multiprocessors," *IBM J. Res. Develop.*, vol. 36, pp. 1001–1013, November 1992.

[5] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky, "Adaptive parallelism and Piranha," *IEEE Computer*, vol. 28, pp. 40–49, Jan. 1995.

[6] R. D. Blumofe and D. S. Park, "Scheduling large-scale parallel computations on networks of workstations," in *Proc. 3rd Int'l Symp. High-Performance Dist. Comput.*, pp. 96–105, August 1994.

[7] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, pp. 57–69, November 1980.

[8] T. Takano, T. Yamada, K. Shutoh, and N. Kanekawa, "Fault-tolerant experiments of the "Hiten" onboard space computer," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 26–33, June 1991.

Table 1: Reconfiguration times for multiple *isolated* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	2
2	3
3	3
4	4
5	4
6	5

Table 2: Reconfiguration times for multiple *chained* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	N/A
2	5
3	15
4	17
5	21
6	28

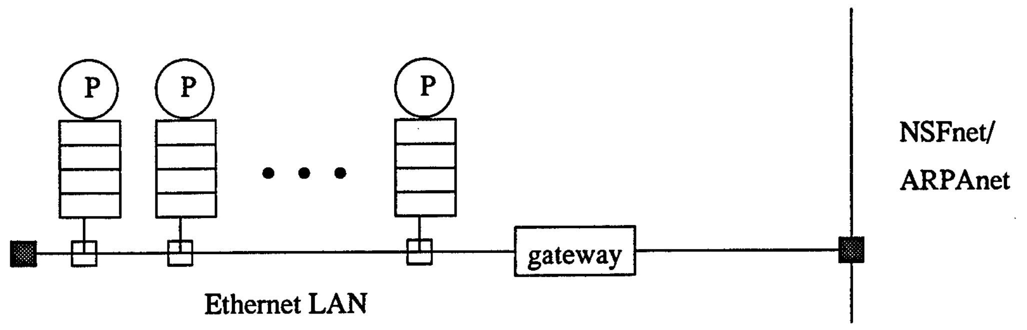

Figure 1: Configuration of the ANTS experimental system.

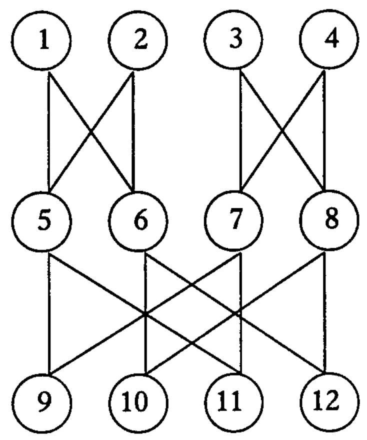

Figure 2: Data dependency graph of an example job.

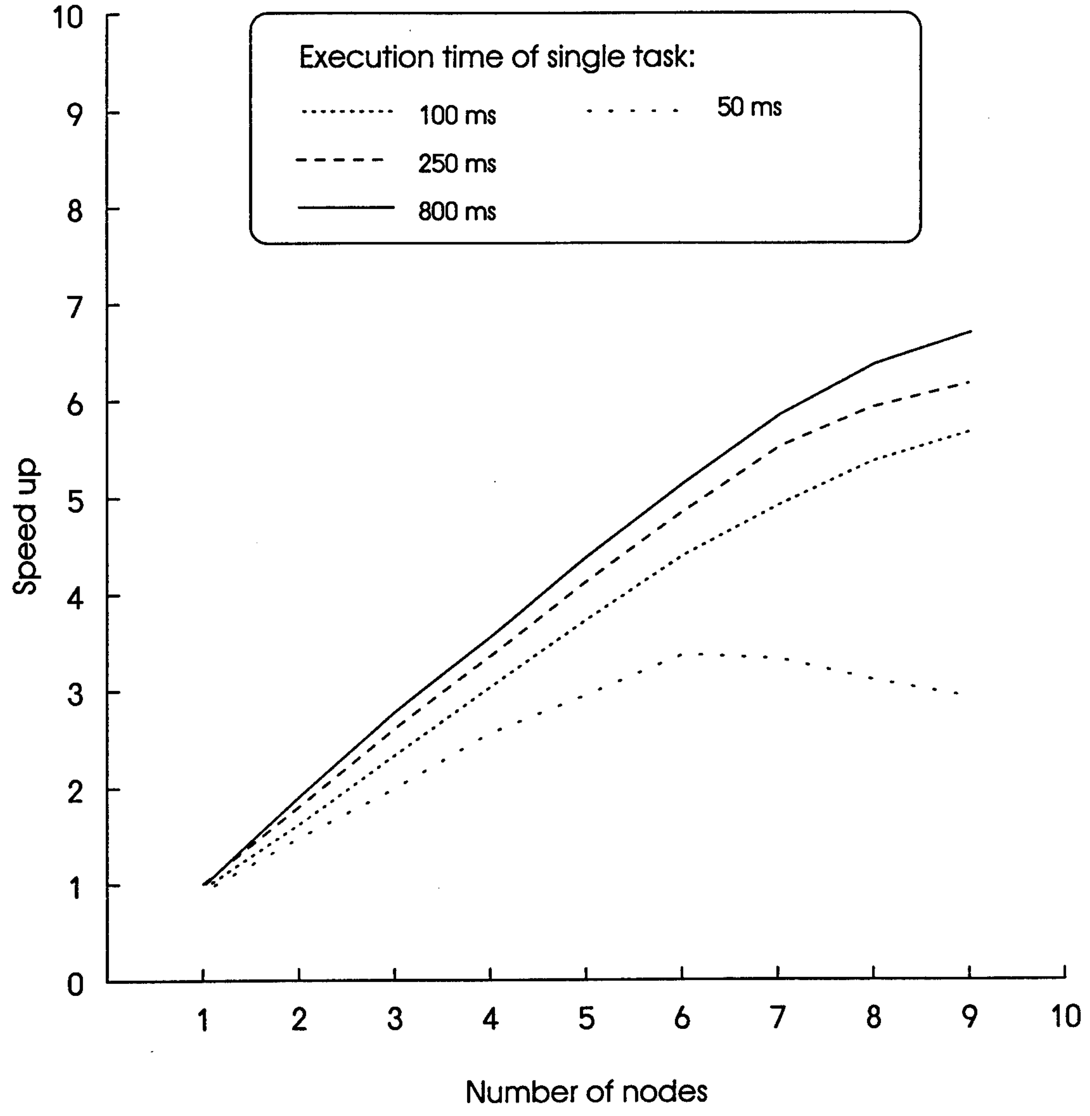

Figure 3: Speed up factors of ANTS with fault tolerant features disabled.

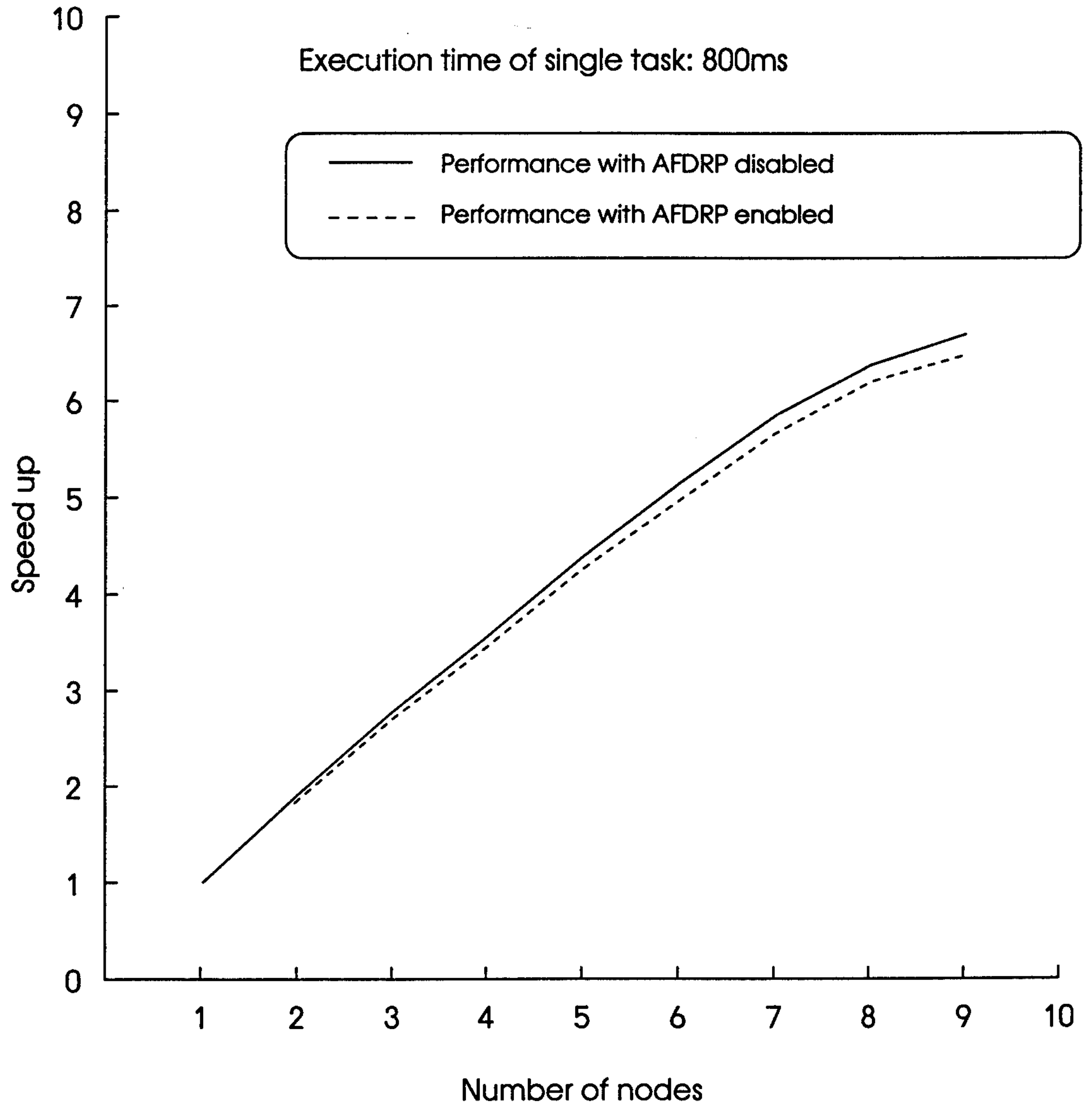

Figure 4: Trade-off in ANTS performance with fault tolerance features.